Authorizing and Directing Configuration Updates in Contemporary IT Infrastructures

Bart Vanbrabant
DistriNet
Dept. of Computer Science,
K.U.Leuven
Celestijnenlaan 200A
Leuven, Belgium
bart.vanbrabant@cs.kuleuven.be

Thomas Delaet
DistriNet
Dept. of Computer Science,
K.U.Leuven
Celestijnenlaan 200A
Leuven, Belgium
thomas.delaet@cs.kuleuven.be

Wouter Joosen
DistriNet
Dept. of Computer Science,
K.U.Leuven
Celestijnenlaan 200A
Leuven, Belgium
wouter.joosen@cs.kuleuven.be

ABSTRACT

All security and non-security equipment in a IT infrastructure has to be consistent with the configuration of the entire IT infrastructure. System management tools are used to manage contemporary IT infrastructures in an efficient and secure manner, and ensure its configuration is consistent and correct. System configuration tools achieve this by using a central configuration model from which all configuration is derived. The central configuration model determines the configuration of the infrastructure and needs to be protected against unauthorised access and changes. In large IT infrastructures there are multiple administrators. Each manages an aspect of the infrastructure and thus requires access to the central model. We propose an approach that enforces access control on the changes that are made to the configuration model. Our approach also includes a method to enforce complex authorisation workflows on configuration model updates in federated infrastructures. We developed a prototype that transforms low level textual updates, to updates to the model. This transformation enables access control at the same abstraction level as the configuration model. The first results of this work have been evaluated and published. In this position paper we argue for further research on securing configuration models and applying access control on updates to the configuration model.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection—*Access control, Authentication, Information flow controls*; C.2.3 [**Computer - Communication Networks**]: Network Operations—*Network management*; C.2.4 [**Computer - Communication Networks**]: Distributed Systems—*Distributed applications*

General Terms

Security

SafeConfig'10, October 4, 2010, Chicago, Illinois, USA.

Keywords

access control, authorisation, systems management, federation

1. INTRODUCTION

A contemporary IT infrastructure consists of hundreds or thousands of devices. Devices can be security appliances such as firewalls or authentication servers, network equipment, all types of servers and end-user desktops and laptops. All these devices need to be configured to function as one infrastructure. To ensure the infrastructure works correctly and is secure, the configuration of each device needs to be consistent with the intended configuration of the infrastructure. Responsibilities and expertise in system administration teams are also not mapped on devices but on subsystems that span the entire infrastructure. For example logging or DNS which needs to be configured on each machine. This means that every system administrator requires access to all devices.

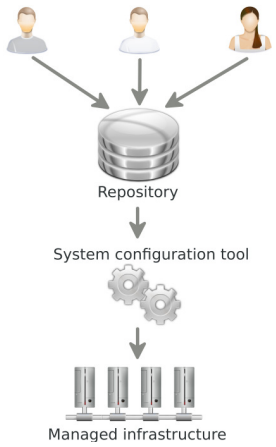

Figure 1: System configuration tool architecture.

System management tools try to solve the configuration problem by deriving the configuration for each device in the infrastructure from a central configuration model [1]. This model is stored in a repository and is modified by the system administration team, as shown in Figure 1. This model contains the desired configuration of the infrastructure. A software agent responsible for the management of a device,

compares the current configuration of the device with the desired configuration and makes any necessary changes [4]. The central configuration model eliminates redundancies in the configuration specification, ensuring the configuration of each device is consistent with the configuration of the entire infrastructure. For example, the hostname of the logserver has to be represented only once in the model, although each device required to use that logserver.

Changes to the central configuration model can result in changes made to all devices in the infrastructure. Therefore access to the configuration model implies control over the infrastructure. In small infrastructures with only a handful system administrators all or nothing access control and authorisation can be sufficient. In large infrastructure with large teams of system administrators or possibly federated administration fine grained access control is required.

In this position paper we argue for further research in securing a configuration model in large infrastructures with complex federated administration. System management tools are necessary to efficiently manage a large infrastructure and to make sure that all devices comply to the policy. This approach requires a central model that needs to be accessible to a lot of people, but requires tight access control because unauthorised changes can have serious consequences. To manage infrastructures in a cost effective and efficient manner, system management tools that use a central configuration model are required. But without further advancement in the state of the art of access control for system management tools, implementing them will introduce serious security threats because of this central configuration model.

Many system management tools use a textual input in the form of source code to represent the configuration model [9]. The current state of the art in access control on this input uses file and directory permissions from the operating system or the repository where the source input is stored. We developed a language agnostic approach that analyses the changes made to the textual representation of the configuration model and transforms the changes into operations on the entities that are represented in the configuration model. Because low level text changes are transformed to operations on the configuration model, access control rules are expressed at the same abstraction level as the configuration model. We prototyped this approach, applied it on a simple configuration model and evaluated in two case studies [15].

In the next section we discuss related work on access control in systems management. In section 3 we provide a brief overview of our approach. Future work and a discussion is provided in section 4.

2. RELATED WORK

State of the art of system configuration tools use a declarative model, have various levels of integration with version control systems and different granularity of access control [2,5,8,10,11]. For workflow enforcement of changes very limited work is available [5, 12].

BCFG2 [10], Cfengine [3], LCFG [2] and Puppet [12] all recommend to use version control repositories. Only BCFG2 provides basic integration for version control and it is only used for reporting purposes. Cfengine and Puppet suggest using branches and tags to create multiple environments to support some sort of workflow. For example, for testing, production and development environments. They are not able to enforce a workflow through these environments upon updates.

Devolved Management of Distributed Infrastructure with Quattor [7] describes how several European grid infrastructures manage large distributed infrastructures with sites under different administrative domains. One of the problems with their current system is the inability to enforce fine-grained authorization. They handle this problem by modularizing the configuration specification using namespaces that the compiler enforces in the file name. This allows the version repository to enforce access control on file names. But the specification in one namespace can still access other namespaces, thus bypassing the access control of the version repository.

Machination [11] provides fine-grained access control based on the manipulation primitives of the XML based input language. Although at a higher level than providing access based on the file names, there is still an abstraction gap between the configuration specification and the access control. The manipulation primitives express what can be changed in the XML input and do not directly express what can be changed in the input specification.

PoDIM [8] includes rules to filter statements before they are applied to the network. These rules are specified at the same abstraction level as the source and apply to the statements in the source specification. However, there are no facilities to enforce a workflow. The specification becomes invalid and cannot be deployed after a change is added that depends on a change that is not yet approved.

Most tools rely on the coarse-grained access control available in version control repositories. Some tools, such as Machination [11], provide very fine-grained access control but in function of the XML representation of the configuration model and not the entities in the configuration model. PoDIM [8] offers filtering of statements at the same abstraction level as the specification but lacks integration with workflow enforcement, thus making it hard to use. Cfengine [5] and Puppet [12] do include provisions to use different branches of a version control repository for different stages in deployment in the same configuration server, but cannot enforce workflows on updates between these stages.

3. AUTHORISATION OF CONFIGURATION MODEL UPDATES

The configuration model used by a system management tool is built from an input in the form of textual source code. This source input is stored on a filesystem or in a repository that uses version tracking. Access control and authorisation in the state of the art is based on operations performed on the files and directories. In state of the art system management tools there is often no link between the file path and the parts of the configuration model represented in the file. Version control systems use diff-like algorithms [14] that operate on flat files to generate changes between two versions of a file. Diff algorithms detect changed lines and produce a list of insert and remove line operations. Applying access control on these operations does not make much sense. The operations are highly syntax dependant and there is only a weak link between the insert and remove operations and the configuration model.

In large infrastructures updates are never applied directly to the production infrastructure. Depending on the contents

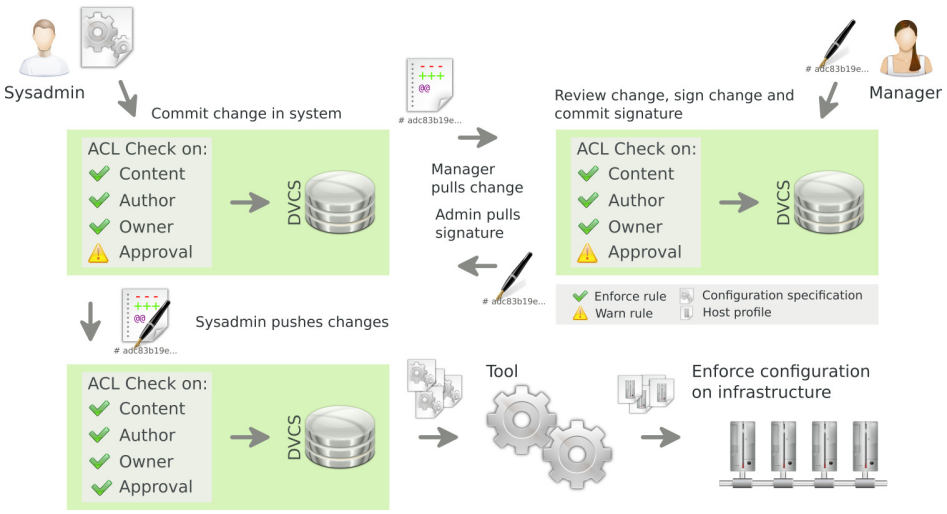

Figure 2: Updating the configuration model using access control.

of the update or the person that produced the update, different authorisations can be required. For example: 1. all changes from junior administrators need to be reviewed and approved by a senior administrator 2. the scenario in Figure 2 where a change needs to be approved by a manager 3. all changes to the production infrastructure outside maintenance windows require approval by two managers 4. in a federated infrastructure changes to the backbone network need to be approved by the management of each of the administrative domains Existing system management tools and access control solutions provide no support for these complex workflows that need to be enforced on model updates.

Our approach [15] transforms the updates on the configuration model by comparing the current and the new version of the input source. It compiles the two versions to an abstract syntax tree [13]. An *edit script* is generated that transforms the old AST to the AST of the new version [6]. This process is represented in Figure 3. Because we are working on the AST, we know the semantics of changes made to the nodes in the abstract syntax tree. Therefore the edit script can be transformed to operations on entities that exist in the configuration model. With this approach, access control rules can be expressed in function of operations on the entities in the configuration model.

For audit purposes the configuration model is often stored in version controlled repositories. These repositories record the change, the user that made the change and a possible log message. In our prototype changes to this repository are digitally signed with the private key of the administrator. During generation of the edit script and the transformation of the edit script, the owner of each entity and the author of each change is tracked. The owner of an entity is the user that added or modified the entity. This ownership and author information is also exposed to the access control rules.

We enforce update workflows by using distributed version control repositories. Each system administrator that makes changes to the configuration model has his own repository. Distributed version control repositories assign a unique identifier to each change based on the contents of the change. To support enforcing of update workflows, a change is authorised by the owner of a key by signing this unique identifier and including it as an update in the repository. Access control rules can include the authorisation of a third party before an update is allowed. Because each distributed version control repository can have its own set of access control rules, very flexible update workflows can be enforced.

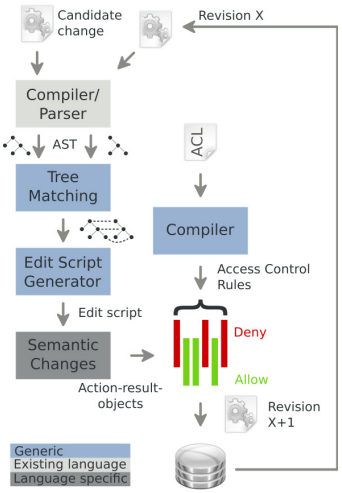

Figure 3: The access control component and the different stages during access control checks.

Figure 2 represents a possible scenario supported by our prototype [15]. A system administrator makes a change that is allowed in his repository but it requires a approval by a manager to push the change into the repository for the production infrastructure. The sysadmin requests the manager to review his change. The manager reviews the change and approves it by signing the identifier of the change. The sysadmin can now push his change to the production repository together with the signature of the manager.

4. FUTURE WORK

The main focus of our future work is on an access control language that is both expressive enough and powerful enough. In our prototype [15] we transformed model updates into operations on configuration model. This results in operations to apply access control on entities that are at the same abstraction level as the entities in the configuration model. We used an access control language that matches the operations on the configuration model using regular expressions. Because the access control language is not integrated with the configuration model it can not take full advantage of the higher abstraction operations. For example using the typing information of the entities that are operated upon. This results in very complex access control rules in our prototype. We think a more powerful access control language is also based on pattern matching but with better integration with the configuration model.

Because the focus of this work was on transforming the changes to a higher abstraction level and enforcing update workflows, we did not evaluate existing access control languages and access control models. In our prototype we applied a simple role based access control model. Further research is required to evaluate existing access control languages and models in a system management context.

We validated our approach on a simple configuration modelling language. When the language becomes more complex with more language constructs, transforming the edit scripts will require priority rules. An abstract syntax tree does not represent all relations that exist in the language because it only models a parent-child and sibling relation. Two operations on the configuration model generate two sets of changes on the AST's that are possibly not disjunct. When the edit script is transformed this could result in other operations on the configuration model than the one that was actually performed. A graph can contain more relations than an AST. Using existing graph matching algorithms edit scripts that are less ambiguous can be generated.

5. CONCLUSION

System management tools are required to manage contemporary IT infrastructures in an efficient and secure manner. These tools use a configuration model that is used to derive all configuration in the infrastructure. The current state of the art does not provide sufficient access control and authorization mechanisms to deploy system management tools without introducing a serious security threat. We provide a first solution and a prototype that addresses some of these issues but further research is required. As a consequence we argue for increased focus on securing configuration models used in system management tools.

6. ACKNOWLEDGEMENTS

Agency for Innovation by Science and Technology in Flanders (IWT)

7. REFERENCES

[1] P. Anderson. Towards a high-level machine configuration system. In *LISA '94: Proceedings of the 8th USENIX conference on System administration*, page 19–26, Berkeley, CA, USA, 1993. USENIX Association, USENIX Association.

[2] P. Anderson. LCFG: A large scale UNIX configuration system. `http://www.lcfg.org`, 2008.

[3] M. Burgess. Cfengine: a site configuration engine. *USENIX Computing Systems*, 8(3):309–402, 1995.

[4] M. Burgess. Computer Immunology. In *LISA '98: Proceedings of the 12th USENIX conference on System administration*, pages 283–298, Berkeley, CA, USA, 1998. USENIX Association.

[5] M. Burgess. Cfengine Website. `http://www.cfengine.org`, 2009.

[6] S. S. Chawathe and H. Garcia-Molina. Meaningful change detection in structured data. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data - SIGMOD 97 SIGMOD 97*, pages 26–37, New York, NY, USA, 1997. ACM.

[7] S. Childs, M. E. Poleggi, C. Loomis, L. F. M. Mejías, M. Jouvin, R. Starink, S. De Weirdt, and G. C. Meliá. Devolved Management of Distributed Infrastructures With Quattor. In *Proceedings of the 22nd Large Installation System Administration (LISA) Conference*, page 175–189, Berkeley, CA, USA, 2008. USENIX Association.

[8] T. Delaet and W. Joosen. PoDIM: A language for high–level configuration management. In *Proceedings of the 21st Large Installation System Administration (LISA) Conference*, pages 1–13, Berkeley, CA, USA, 2007. USENIX Association.

[9] T. Delaet, W. Joosen, and B. Vanbrabant. A survey of system configuration tools. In *Proceedings of the 24th Large Installations Systems Administration (LISA) conference*, San Jose, CA, USA, 11/2010 2010. Usenix Association, Usenix Association.

[10] N. Desai. Bcfg2: A Pay as You Go Approach to Configuration Complexity. 2005.

[11] C. Higgs. Authorisation and Delegation in the Machination Configuration System. In *Proceedings of the 22nd Large Installation System Administration (LISA) Conference*, pages 191–199, Berkeley, CA, USA, 2008. USENIX Association.

[12] L. Kanies. Puppet Website. `http://reductivelabs.com/projects/puppet/`, 2008.

[13] J. McCarthy. Towards a mathematical science of computation. *Information Processing*, 62:21–28, 1962.

[14] E. W. Myers. An O(ND) difference algorithm and its variations. *Algorithmica*, 1(1):251–266, 1986.

[15] B. Vanbrabant, T. Delaet, and W. Joosen. Federated access control and workflow enforcement in systems configuration. In *Proceedings of the 23rd Large Installations Systems Administration (LISA) conference*, page 129–143, Baltimore, MD, USA, 11/2009 2009. Usenix Association, Usenix Association.